Technical Disclosure Commons

Defensive Publications Series

December 07, 2017

Machine-Learned Caching of Datasets

Etienne J. Membrives

Machine-Learned Caching of Datasets

Inventor: Etienne J. Membrives

Overview

Generally, the present disclosure is directed to creating and/or modifying a pre-cache for a client device connected to a remote server containing a dataset. In particular, in some implementations, the systems and methods of the present disclosure can include or otherwise leverage one or more machine-learned models to predict the likelihood a particular piece of data will be used (e.g. opened, edited, saved, etc.) within a time frame based on information about the data, the user's interaction with the data, and/or the user's schedule.

Example Figures

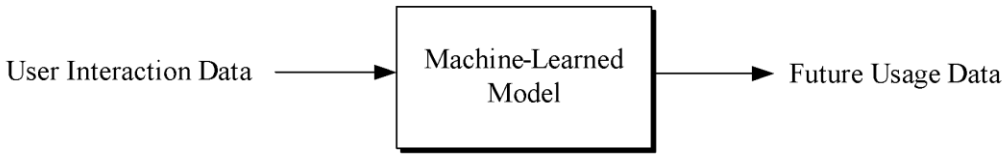

Figure 1

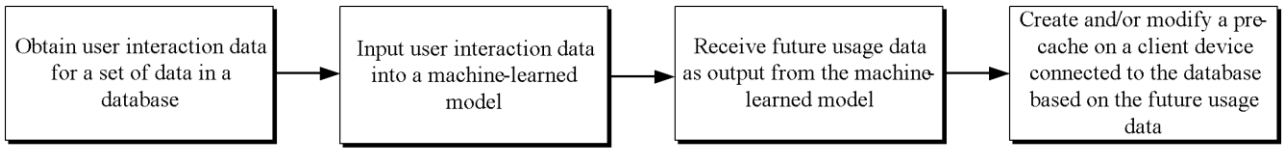

Figure 6

Introduction

With the increased availability of the Internet (e.g. through faster and broader scope of wired and wireless networks, Wi-Fi, cellular phone data, broadband data, etc.), it has become practical to store datasets (e.g. a collection of documents, photos, emails, etc.) on a remote server (e.g. the "cloud") typically accessible via one or more client devices (e.g. a mobile phone, tablet, laptop, etc.). For instance, many users store a majority of a dataset on a remote server with only a relatively small portion, if any, of the dataset being saved locally, i.e. on the client device. The client devices can also contain a user-configurable "pre-cache" of frequently accessed data within the dataset, such as a frequently used document or picture. The pre-cache speeds up the operation of the client device, such as by saving the device from downloading the data every time it is opened and additionally enables viewing of the data when the client device is not connected to the remote server. These pre-caches are typically configured manually by the user of the client device, which can be a tedious process. What is needed is a system for creating and/or modifying a pre-cache dynamically based on the user's interaction with the dataset and/or the user's schedule.

Summary

Generally, the present disclosure is directed to creating and/or modifying a pre-cache for a client device connected to a remote server containing a dataset. In particular, in some implementations, the systems and methods of the present disclosure can include or otherwise leverage one or more machine-learned models to predict the likelihood a particular piece of data will be used (e.g. opened, edited, saved, etc.) within a time frame based on information about the

data, the user's interaction with the data, and/or the user's schedule.

A computing system can create and/or modify a pre-cache on a client device to store a portion of a remotely-stored dataset on the client device. For instance, the portion of the remotely-stored dataset can be a portion of the dataset that a user of the client device is likely to need with high priority, is frequently used (e.g. opened, edited, saved) by the user, or has some other special characteristic. In particular, the system can employ a machine-learned model to predict the likelihood a particular piece of data will be used (e.g. opened, edited, saved, etc.) within a time frame based on information about the data, user interaction with the data, and/or scheduling information. The machine-learned model can be trained based on data associated with existing datasets. The machine-learned model can be a neural network, such as a "wide & deep" neural network. The system can use the prediction (e.g. by a "caching agent" within the system) to determine and download the data that the user is likely to need (e.g. frequently use or need access to while offline) to the pre-cache. The system can additionally and/or alternatively remove data from the pre-cache that is not needed.

The machine-learned model can provide a prediction based on information about the data (e.g. word/pixel contents, title, contents of links embedded within the document, metadata associated with the document, etc.), interaction with the data (e.g. the last edited date, the last viewed date, the created date, frequency of use, etc.), and/or scheduling information (e.g. the user's current location, day of week, day of year, user's future locations, etc.).

For instance, the model can provide a prediction based on information about the data such as the textual and/or pixel contents of the data (e.g. the contents of a text document or

picture), the title of the data, the contents of links within the data (e.g. if the links are to websites, other documents, pictures, etc.), metadata associated with the data (e.g. any additional tags, markings, or other information that can be associated with the data). For example, if the data is an email, the metadata may include the author, send date, recipients, etc. As another example, if the data is an image, the metadata may include the date taken, camera used, comments or annotations, etc. As yet another example, if the data is a text document, the metadata may include author(s), date written, program used, etc.

The machine-learned model can additionally and/or alternatively provide a prediction based on information about interaction with the data. For instance, the information can include creation date, last edited date, last opened date, frequency and/or number of times opened and/or edited, and/or other suitable characteristics of user interaction. Additionally and/or alternatively, the information can include the user and/or the device that performed the interaction. The information can be determined for a client device (e.g. identified via serial code, IP address, MAC address, and/or other unique characteristic and/or combination of characteristics) and/or for a user of one or more client devices (e.g. identified via name, username, email account, or other suitable identifier associated with a user). A single user may be associated with a single client device or multiple client devices, and the model may provide a prediction based on the user's interaction on only one client device (e.g. the device for which the pre-cache is being predicted) or on several (e.g. all) of the user's associated client devices. In some embodiments, the client device may have several users, and the model may provide a prediction based on the interactions performed on the device for one user (e.g. if separate pre-caches are being generated

on the client device for each user, the model may only consider the interactions from the user for which the pre-cache is being generated) or several (e.g. all) of the device's users (e.g. for a unified pre-cache across multiple users). If the dataset is shared among multiple users, the model may provide a prediction based on user interactions for only the user and/or device for which the pre-cache is being created or modified and/or from multiple users and/or multiple devices. In some embodiments, the source for information provided to the model (e.g. from one or several devices, users, etc.) may be configurable within the system (e.g. by a virtual and/or physical control panel). In some embodiments, the source for information provided to the model may be configurable for all users and/or devices or on a per-user and/or per-device basis.

The machine-learned model can additionally and/or alternatively use information about scheduling. For example, a schedule can be determined for a client device (e.g. use case and/or location of the device) and/or for a user (e.g. when the user will use a device and/or which devices the user will use and/or location of the user). The machine-learned model can use information such as the user's current location, day of the week, day of the year, and/or the future locations of the user. The future locations of the user can be determined (e.g. by an algorithm and/or the same and/or another machine-learned model) from, for example, the user's agenda, location history, prompts provided to the user, and/or other suitable data.

The machine-learned model can output a prediction of the likelihood that a particular piece of data will be opened within a time period. For instance, the machine-learned model can predict the likelihood the data will be opened within an hour, day, week, month, year, or other suitable time interval. The time interval can be determined by the machine-learned model (e.g.

from scheduling information). For example, the time interval may be based on time the client device spends offline, or may be based on usage of the client device. In some embodiments, the model may predict the amount of time before the data is opened again (e.g. the longer time corresponds to lower probability and vice versa). For example, if the data is opened ten times an hour on average, the model may predict the data will be opened within 6 minutes, or 0.1 hours, or similar. In some embodiments, the prediction may be refined further, e.g. the average is generalized to only a workday or other time of operation of the client device. Other considerations, such as the importance of the data (e.g. determined by metadata, contents, interactions, etc.) can be used in the prediction.

Based on the prediction from the machine-learned model, the system may determine data to be downloaded to the pre-cache. The pre-cache can conceivably have limited space, so the system may download the data with the predicted most frequent use, most importance, or other metric. The system can also delete data from the pre-cache that is determined to be unnecessary (e.g., if the pre-cache is full, the data may have lower importance than some new data which must be added, or interactions with the data may have decreased over time).

Further to the descriptions above, a user may be provided with controls allowing the user to make an election as to both if and when systems, programs or features described herein may enable collection of user information (e.g., information about a user's interaction with data, a user's schedule, a user's preferences, or a user's current location), and if the user is sent content or communications from a server. In addition, certain data may be treated in one or more ways before it is stored or used, so that personally identifiable information is removed. For example, a

user's identity may be treated so that no personally identifiable information can be determined for the user, or a user's geographic location may be generalized where location information is obtained (such as to a city, ZIP code, or state level), so that a particular location of a user cannot be determined. Thus, the user may have control over what information is collected about the user, how that information is used, and what information is provided to the user.

Thus, the system can determine an optimal pre-cache for a client device based on one or more users' interaction with data within a dataset. This can enable the users to store data on a remote server with increased confidence that necessary data will be available locally when needed and/or can increase efficiency of using the client device. Additionally, this can save the users the hassle and potential human error of manually creating a pre-cache for the client device, and can enable pre-caches tailored automatically to an individual user.

Detailed Description

As described above, the present disclosure is directed to creating and/or modifying a pre-cache for a client device connected to a remote server containing a dataset. In particular, in some implementations, the systems and methods of the present disclosure can include or otherwise leverage one or more machine-learned models to predict the likelihood a particular piece of data will be used (e.g. opened, edited, saved, etc.) within a time frame based on information about the data, the user's interaction with the data, and/or the user's schedule.

Figure 1 depicts a block diagram of an example machine-learned model according to example implementations of the present disclosure. As illustrated in Figure 1, in some implementations, the machine-learned model is trained to receive input data of one or more types

and, in response, provide output data of one or more types. Thus, Figure 1 illustrates the machine-learned model performing inference.

In some implementations, the input data can include one or more features that are associated with an instance or an example. In some implementations, the one or more features associated with the instance or example can be organized into a feature vector. In some implementations, the output data can include one or more predictions. Predictions can also be referred to as inferences. Thus, given features associated with a particular instance, the machine-learned model can output a prediction for such instance based on the features.

The machine-learned model can be or include one or more of various different types of machine-learned models. In particular, in some implementations, the machine-learned model can perform classification, regression, clustering, anomaly detection, recommendation generation, and/or other tasks.

In some implementations, the machine-learned model can perform various types of classification based on the input data. For example, the machine-learned model can perform binary classification or multiclass classification. In binary classification, the output data can include a classification of the input data into one of two different classes. In multiclass classification, the output data can include a classification of the input data into one (or more) of more than two classes. The classifications can be single label or multi-label.

In some implementations, the machine-learned model can perform discrete categorical classification in which the input data is simply classified into one or more classes or categories.

In some implementations, the machine-learned model can perform classification in which

the machine-learned model provides, for each of one or more classes, a numerical value descriptive of a degree to which it is believed that the input data should be classified into the corresponding class. In some instances, the numerical values provided by the machine-learned model can be referred to as "confidence scores" that are indicative of a respective confidence associated with classification of the input into the respective class. In some implementations, the confidence scores can be compared to one or more thresholds to render a discrete categorical prediction. In some implementations, only a certain number of classes (e.g., one) with the relatively largest confidence scores can be selected to render a discrete categorical prediction.

In some implementations, the machine-learned model can provide a probabilistic classification. For example, the machine-learned model can be able to predict, given a sample input, a probability distribution over a set of classes. Thus, rather than outputting only the most likely class to which the sample input should belong, the machine-learned model can output, for each class, a probability that the sample input belongs to such class. In some implementations, the probability distribution over all possible classes can sum to one. In some implementations, a softmax function or layer can be used to squash a set of real values respectively associated with the possible classes to a set of real values in the range (0, 1) that sum to one.

In some implementations, the probabilities provided by the probability distribution can be compared to one or more thresholds to render a discrete categorical prediction. In some implementations, only a certain number of classes (e.g., one) with the relatively largest predicted probability can be selected to render a discrete categorical prediction.

In some implementations in which the machine-learned model performs classification,

the machine-learned model can be trained using supervised learning techniques. For example, the machine-learned model can be trained on a training dataset that includes training examples labeled as belonging (or not belonging) to one or more classes. Further details regarding supervised training techniques are provided below.

In some implementations, the machine-learned model can perform regression to provide output data in the form of a continuous numeric value. The continuous numeric value can correspond to any number of different metrics or numeric representations, including, for example, currency values, scores, or other numeric representations. As examples, the machine-learned model can perform linear regression, polynomial regression, or nonlinear regression. As examples, the machine-learned model can perform simple regression or multiple regression. As described above, in some implementations, a softmax function or layer can be used to squash a set of real values respectively associated with a two or more possible classes to a set of real values in the range (0, 1) that sum to one.

In some implementations, the machine-learned model can perform various types of clustering. For example, the machine-learned model can identify one or more previously-defined clusters to which the input data most likely corresponds. As another example, the machine-learned model can identify one or more clusters within the input data. That is, in instances in which the input data includes multiple objects, documents, or other entities, the machine-learned model can sort the multiple entities included in the input data into a number of clusters. In some implementations in which the machine-learned model performs clustering, the machine-learned model can be trained using unsupervised learning techniques.

In some implementations, the machine-learned model can perform anomaly detection or outlier detection. For example, the machine-learned model can identify input data that does not conform to an expected pattern or other characteristic (e.g., as previously observed from previous input data). As examples, the anomaly detection can be used for fraud detection or system failure detection.

In some implementations, the machine-learned model can provide output data in the form of one or more recommendations. For example, the machine-learned model can be included in a recommendation system or engine. As an example, given input data that describes previous outcomes for certain entities (e.g., a score, ranking, or rating indicative of an amount of success or enjoyment), the machine-learned model can output a suggestion or recommendation of one or more additional entities that, based on the previous outcomes, are expected to have a desired outcome (e.g., elicit a score, ranking, or rating indicative of success or enjoyment). As one example, given input data descriptive of a number of products purchased or rated highly by a user, a recommendation system can output a suggestion or recommendation of an additional product that the user might enjoy or wish to purchase.

In some implementations, the machine-learned model can act as an agent within an environment. For example, the machine-learned model can be trained using reinforcement learning, which will be discussed in further detail below.

In some implementations, the machine-learned model can be a parametric model while, in other implementations, the machine-learned model can be a non-parametric model. In some implementations, the machine-learned model can be a linear model while, in other

implementations, the machine-learned model can be a non-linear model.

As described above, the machine-learned model can be or include one or more of various different types of machine-learned models. Examples of such different types of machine-learned models are provided below for illustration. One or more of the example models described below can be used (e.g., combined) to provide the output data in response to the input data. Additional models beyond the example models provided below can be used as well.

In some implementations, the machine-learned model can be or include one or more classifier models such as, for example, linear classification models; quadratic classification models; etc.

In some implementations, the machine-learned model can be or include one or more regression models such as, for example, simple linear regression models; multiple linear regression models; logistic regression models; stepwise regression models; multivariate adaptive regression splines; locally estimated scatterplot smoothing models; etc.

In some implementations, the machine-learned model can be or include one or more decision tree-based models such as, for example, classification and/or regression trees; iterative dichotomiser 3 decision trees; C4.5 decision trees; chi-squared automatic interaction detection decision trees; decision stumps; conditional decision trees; etc.

In some implementations, the machine-learned model can be or include one or more kernel machines. In some implementations, the machine-learned model can be or include one or more support vector machines.

In some implementations, the machine-learned model can be or include one or more

instance-based learning models such as, for example, learning vector quantization models; self-organizing map models; locally weighted learning models; etc.

In some implementations, the machine-learned model can be or include one or more nearest neighbor models such as, for example, k-nearest neighbor classifications models; k-nearest neighbors regression models; etc.

In some implementations, the machine-learned model can be or include one or more Bayesian models such as, for example, naïve Bayes models; Gaussian naïve Bayes models; multinomial naïve Bayes models; averaged one-dependence estimators; Bayesian networks; Bayesian belief networks; hidden Markov models; etc.

In some implementations, the machine-learned model can be or include one or more artificial neural networks (also referred to simply as neural networks). A neural network can include a group of connected nodes, which also can be referred to as neurons or perceptrons. A neural network can be organized into one or more layers. Neural networks that include multiple layers can be referred to as "deep" networks. A deep network can include an input layer, an output layer, and one or more hidden layers positioned between the input layer and the output layer. The nodes of the neural network can be connected or non-fully connected.

In some implementations, the machine-learned model can be or include one or more feed forward neural networks. In feed forward networks, the connections between nodes do not form a cycle. For example, each connection can connect a node from an earlier layer to a node from a later layer.

In some instances, the machine-learned model can be or include one or more recurrent

neural networks. In some instances, at least some of the nodes of a recurrent neural network can form a cycle. Recurrent neural networks can be especially useful for processing input data that is sequential in nature. In particular, in some instances, a recurrent neural network can pass or retain information from a previous portion of the input data sequence to a subsequent portion of the input data sequence through the use of recurrent or directed cyclical node connections.

As one example, sequential input data can include time-series data (e.g., sensor data versus time or imagery captured at different times). For example, a recurrent neural network can analyze sensor data versus time to detect or predict a swipe direction, to perform handwriting recognition, etc. As another example, sequential input data can include words in a sentence (e.g., for natural language processing, speech detection or processing, etc.); notes in a musical composition; sequential actions taken by a user (e.g., to detect or predict sequential application usage); sequential object states; etc.

Example recurrent neural networks include long short-term (LSTM) recurrent neural networks; gated recurrent units; bi-direction recurrent neural networks; continuous time recurrent neural networks; neural history compressors; echo state networks; Elman networks; Jordan networks; recursive neural networks; Hopfield networks; fully recurrent networks; sequence-to-sequence configurations; etc.

In some implementations, the machine-learned model can be or include one or more convolutional neural networks. In some instances, a convolutional neural network can include one or more convolutional layers that perform convolutions over input data using learned filters. Filters can also be referred to as kernels. Convolutional neural networks can be especially useful

for vision problems such as when the input data includes imagery such as still images or video. However, convolutional neural networks can also be applied for natural language processing.

In some implementations, the machine-learned model can be or include one or more generative networks such as, for example, generative adversarial networks. Generative networks can be used to generate new data such as new images or other content.

In some implementations, the machine-learned model can be or include an autoencoder. In some instances, the aim of an autoencoder is to learn a representation (e.g., a lower-dimensional encoding) for a set of data, typically for the purpose of dimensionality reduction. For example, in some instances, an autoencoder can seek to encode the input data and then provide output data that reconstructs the input data from the encoding. Recently, the autoencoder concept has become more widely used for learning generative models of data. In some instances, the autoencoder can include additional losses beyond reconstructing the input data.

In some implementations, the machine-learned model can be or include one or more other forms of artificial neural networks such as, for example, deep Boltzmann machines; deep belief networks; stacked autoencoders; etc. Any of the neural networks described herein can be combined (e.g., stacked) to form more complex networks.

In some implementations, one or more neural networks can be used to provide an embedding based on the input data. For example, the embedding can be a representation of knowledge abstracted from the input data into one or more learned dimensions. In some instances, embeddings can be a useful source for identifying related entities. In some instances embeddings can be extracted from the output of the network, while in other instances

embeddings can be extracted from any hidden node or layer of the network (e.g., a close to final but not final layer of the network). Embeddings can be useful for performing auto suggest next video, product suggestion, entity or object recognition, etc. In some instances, embeddings be useful inputs for downstream models. For example, embeddings can be useful to generalize input data (e.g., search queries) for a downstream model or processing system.

In some implementations, the machine-learned model can include one or more clustering models such as, for example, k-means clustering models; k-medians clustering models; expectation maximization models; hierarchical clustering models; etc.

In some implementations, the machine-learned model can perform one or more dimensionality reduction techniques such as, for example, principal component analysis; kernel principal component analysis; graph-based kernel principal component analysis; principal component regression; partial least squares regression; Sammon mapping; multidimensional scaling; projection pursuit; linear discriminant analysis; mixture discriminant analysis; quadratic discriminant analysis; generalized discriminant analysis; flexible discriminant analysis; autoencoding; etc.

In some implementations, the machine-learned model can perform or be subjected to one or more reinforcement learning techniques such as Markov decision processes; dynamic programming; Q functions or Q-learning; value function approaches; deep Q-networks; differentiable neural computers; asynchronous advantage actor-critics; deterministic policy gradient; etc.

In some implementations, the machine-learned model can be an autoregressive model. In

some instances, an autoregressive model can specify that the output data depends linearly on its own previous values and on a stochastic term. In some instances, an autoregressive model can take the form of a stochastic difference equation. One example autoregressive model is WaveNet, which is a generative model for raw audio.

In some implementations, the machine-learned model can include or form part of a multiple model ensemble. As one example, bootstrap aggregating can be performed, which can also be referred to as "bagging." In bootstrap aggregating, a training dataset is split into a number of subsets (e.g., through random sampling with replacement) and a plurality of models are respectively trained on the number of subsets. At inference time, respective outputs of the plurality of models can be combined (e.g., through averaging, voting, or other techniques) and used as the output of the ensemble.

One example model ensemble is a random forest, which can also be referred to as a random decision forest. Random forests are an ensemble learning method for classification, regression, and other tasks. Random forests are generated by producing a plurality of decision trees at training time. In some instances, at inference time, the class that is the mode of the classes (classification) or the mean prediction (regression) of the individual trees can be used as the output of the forest. Random decision forests can correct for decision trees' tendency to overfit their training set.

Another example ensemble technique is stacking, which can, in some instances, be referred to as stacked generalization. Stacking includes training a combiner model to blend or otherwise combine the predictions of several other machine-learned models. Thus, a plurality of

machine-learned models (e.g., of same or different type) can be trained based on training data. In addition, a combiner model can be trained to take the predictions from the other machine-learned models as inputs and, in response, produce a final inference or prediction. In some instances, a single-layer logistic regression model can be used as the combiner model.

Another example ensemble technique is boosting. Boosting can include incrementally building an ensemble by iteratively training weak models and then adding to a final strong model. For example, in some instances, each new model can be trained to emphasize the training examples that previous models misinterpreted (e.g., misclassified). For example, a weight associated with each of such misinterpreted examples can be increased. One common implementation of boosting is AdaBoost, which can also be referred to as Adaptive Boosting. Other example boosting techniques include LPBoost; TotalBoost; BrownBoost; xgboost; MadaBoost, LogitBoost, gradient boosting; etc.

Furthermore, any of the models described above (e.g., regression models and artificial neural networks) can be combined to form an ensemble. As an example, an ensemble can include a top level machine-learned model or a heuristic function to combine and/or weight the outputs of the models that form the ensemble.

In some implementations, multiple machine-learned models (e.g., that form an ensemble) can be linked and trained jointly (e.g., through backpropagation of errors sequentially through the model ensemble). However, in some implementations, only a subset (e.g., one) of the jointly trained models is used for inference.

In some implementations, the machine-learned model can be used to preprocess the input

data for subsequent input into another model. For example, the machine-learned model can perform dimensionality reduction techniques and embeddings (e.g., matrix factorization, principal components analysis, singular value decomposition, word2vec/GLOVE, and/or related approaches); clustering; and even classification and regression for downstream consumption. Many of these techniques have been discussed above and will be further discussed below.

Referring again to Figure 1, and as discussed above, the machine-learned model can be trained or otherwise configured to receive the input data and, in response, provide the output data. The input data can include different types, forms, or variations of input data. As examples, in various implementations, the input data can include information about the data (e.g. word/pixel contents, title, contents of links embedded within the document, metadata associated with the document, etc.), interaction with the data (e.g. the last edited date, the last viewed date, the created date, frequency of use, etc.), and/or scheduling information (e.g. the user's current location, day of week, day of year, user's future locations, etc.).

For instance, the model can provide a prediction based on information about the data such as the textual and/or pixel contents of the data (e.g. the contents of a text document or picture), the title of the data, the contents of links within the data (e.g. if the links are to websites, other documents, pictures, etc.), metadata associated with the data (e.g. any additional tags, markings, or other information that can be associated with the data). For example, if the data is an email, the metadata may include the author, send date, recipients, etc. As another example, if the data is an image, the metadata may include the date taken, camera used, comments or annotations, etc. As yet another example, if the data is a text document, the metadata may

include author(s), date written, program used, etc.

The machine-learned model can additionally and/or alternatively provide a prediction based on information about interaction with the data. For instance, the information can include creation date, last edited date, last opened date, frequency and/or number of times opened and/or edited, and/or other suitable characteristics of user interaction. Additionally and/or alternatively, the information can include the user and/or the device that performed the interaction. The information can be determined for a client device (e.g. identified via serial code, IP address, MAC address, and/or other unique characteristic and/or combination of characteristics) and/or for a user of one or more client devices (e.g. identified via name, username, email account, or other suitable identifier associated with a user). A single user may be associated with a single client device or multiple client devices, and the model may provide a prediction based on the user's interaction on only one client device (e.g. the device for which the pre-cache is being predicted) or on several (e.g. all) of the user's associated client devices. In some embodiments, the client device may have several users, and the model may provide a prediction based on the interactions performed on the device for one user (e.g. if separate pre-caches are being generated on the client device for each user, the model may only consider the interactions from the user for which the pre-cache is being generated) or several (e.g. all) of the device's users (e.g. for a unified pre-cache across multiple users). If the dataset is shared among multiple users, the model may provide a prediction based on user interactions for only the user and/or device for which the pre-cache is being created or modified and/or from multiple users and/or multiple devices. In some embodiments, the source for information provided to the model (e.g. from one or several

devices, users, etc.) may be configurable within the system (e.g. by a virtual and/or physical control panel). In some embodiments, the source for information provided to the model may be configurable for all users and/or devices or on a per-user and/or per-device basis.

The machine-learned model can additionally and/or alternatively use information about scheduling. For example, a schedule can be determined for a client device (e.g. use case and/or location of the device) and/or for a user (e.g. when the user will use a device and/or which devices the user will use and/or location of the user). The machine-learned model can use information such as the user's current location, day of the week, day of the year, and/or the future locations of the user. The future locations of the user can be determined (e.g. by an algorithm and/or the same and/or another machine-learned model) from, for example, the user's agenda, location history, prompts provided to the user, and/or other suitable data.

In some implementations, the machine-learned model can receive and use the input data in its raw form. In some implementations, the raw input data can be preprocessed. Thus, in addition or alternatively to the raw input data, the machine-learned model can receive and use the preprocessed input data.

In some implementations, preprocessing the input data can include extracting one or more additional features from the raw input data. For example, feature extraction techniques can be applied to the input data to generate one or more new, additional features. Example feature extraction techniques include edge detection; corner detection; blob detection; ridge detection; scale-invariant feature transform; motion detection; optical flow; Hough transform; etc.

In some implementations, the extracted features can include or be derived from

transformations of the input data into other domains and/or dimensions. As an example, the extracted features can include or be derived from transformations of the input data into the frequency domain. For example, wavelet transformations and/or fast Fourier transforms can be performed on the input data to generate additional features.

In some implementations, the extracted features can include statistics calculated from the input data or certain portions or dimensions of the input data. Example statistics include the mode, mean, maximum, minimum, or other metrics of the input data or portions thereof.

In some implementations, as described above, the input data can be sequential in nature. In some instances, the sequential input data can be generated by sampling or otherwise segmenting a stream of input data. As one example, frames can be extracted from a video. In some implementations, sequential data can be made non-sequential through summarization.

As another example preprocessing technique, portions of the input data can be imputed. For example, additional synthetic input data can be generated through interpolation and/or extrapolation.

As another example preprocessing technique, some or all of the input data can be scaled, standardized, normalized, generalized, and/or regularized. Example regularization techniques include ridge regression; least absolute shrinkage and selection operator (LASSO); elastic net; least-angle regression; cross-validation; L1 regularization; L2 regularization; etc. As one example, some or all of the input data can be normalized by subtracting the mean across a given dimension’s feature values from each individual feature value and then dividing by the standard deviation or other metric.

As another example preprocessing technique, some or all or the input data can be quantized or discretized. As yet another example, qualitative features or variables included in the input data can be converted to quantitative features or variables. For example, one hot encoding can be performed.

In some implementations, dimensionality reduction techniques can be applied to the input data prior to input into the machine-learned model. Several examples of dimensionality reduction techniques are provided above, including, for example, principal component analysis; kernel principal component analysis; graph-based kernel principal component analysis; principal component regression; partial least squares regression; Sammon mapping; multidimensional scaling; projection pursuit; linear discriminant analysis; mixture discriminant analysis; quadratic discriminant analysis; generalized discriminant analysis; flexible discriminant analysis; autoencoding; etc.

In some implementations, during training, the input data can be intentionally deformed in any number of ways to increase model robustness, generalization, or other qualities. Example techniques to deform the input data include adding noise; changing color, shade, or hue; magnification; segmentation; amplification; etc.

Referring again to Figure 1, in response to receipt of the input data, the machine-learned model can provide the output data. The output data can include different types, forms, or variations of output data. As examples, in various implementations, the output data can include a prediction of the likelihood that a particular piece of data will be opened within a time period. For instance, the machine-learned model can predict the likelihood the data will be opened

within an hour, day, week, month, year, or other suitable time interval. The time interval can be determined by the machine-learned model (e.g. from scheduling information). For example, the time interval may be based on time the client device spends offline, or may be based on usage of the client device. In some embodiments, the model may predict the amount of time before the data is opened again (e.g. the longer time corresponds to lower probability and vice versa). For example, if the data is opened ten times an hour on average, the model may predict the data will be opened within 6 minutes, or 0.1 hours, or similar. In some embodiments, the prediction may be refined further, e.g. the average is generalized to only a workday or other time of operation of the client device. Other considerations, such as the importance of the data (e.g. determined by metadata, contents, interactions, etc.) can be used in the prediction.

As discussed above, in some implementations, the output data can include various types of classification data (e.g., binary classification, multiclass classification, single label, multi-label, discrete classification, regressive classification, probabilistic classification, etc.) or can include various types of regressive data (e.g., linear regression, polynomial regression, nonlinear regression, simple regression, multiple regression, etc.). In other instances, the output data can include clustering data, anomaly detection data, recommendation data, or any of the other forms of output data discussed above.

In some implementations, the output data can influence downstream processes or decision making. As one example, in some implementations, the output data can be interpreted and/or acted upon by a rules-based regulator.

Thus, the present disclosure provides systems and methods that include or otherwise

leverage one or more machine-learned models to predict the likelihood a particular piece of data will be used (e.g. opened, edited, saved, etc.) within a time frame based on information about the data, the user's interaction with the data, and/or the user's schedule. Any of the different types or forms of input data described above can be combined with any of the different types or forms of machine-learned models described above to provide any of the different types or forms of output data described above.

The systems and methods of the present disclosure can be implemented by or otherwise executed on one or more computing devices. Example computing devices include user computing devices (e.g., laptops, desktops, and mobile computing devices such as tablets, smartphones, wearable computing devices, etc.); embedded computing devices (e.g., devices embedded within a vehicle, camera, image sensor, industrial machine, satellite, gaming console or controller, or home appliance such as a refrigerator, thermostat, energy meter, home energy manager, smart home assistant, etc.); server computing devices (e.g., database servers, parameter servers, file servers, mail servers, print servers, web servers, game servers, application servers, etc.); dedicated, specialized model processing or training devices; virtual computing devices; other computing devices or computing infrastructure; or combinations thereof.

Thus, in some implementations, the machine-learned model can be stored at and/or implemented locally by an embedded device or a user computing device such as a mobile device. Output data obtained through local implementation of the machine-learned model at the embedded device or the user computing device can be used to improve performance of the embedded device or the user computing device (e.g., an application implemented by the

embedded device or the user computing device). As one example, Figure 2 illustrates a block diagram of an example computing device that stores and implements a machine-learned model locally.

In other implementations, the machine-learned model can be stored at and/or implemented by a server computing device. In some instances, output data obtained through implementation of the machine-learned model at the server computing device can be used to improve other server tasks or can be used by other non-user devices to improve services performed by or for such other non-user devices. For example, the output data can improve other downstream processes performed by the server computing device for a user computing device or embedded computing device. In other instances, output data obtained through implementation of the machine-learned model at the server computing device can be sent to and used by a user computing device, an embedded computing device, or some other client device. For example, the server computing device can be said to perform machine learning as a service. As one example, Figure 3 illustrates a block diagram of an example client computing device that can communicate over a network with an example server computing system that includes a machine-learned model.

In yet other implementations, different respective portions of the machine-learned model can be stored at and/or implemented by some combination of a user computing device; an embedded computing device; a server computing device; etc.

Computing devices can perform graph processing techniques or other machine learning techniques using one or more machine learning platforms, frameworks, and/or libraries, such as,

for example, TensorFlow, Caffe/Caffe2, Theano, Torch/PyTorch, MXnet, CNTK, etc.

Computing devices can be distributed at different physical locations and connected via one or more networks. Distributed computing devices can operate according to sequential computing architectures, parallel computing architectures, or combinations thereof. In one example, distributed computing devices can be controlled or guided through use of a parameter server.

In some implementations, multiple instances of the machine-learned model can be parallelized to provide increased processing throughput. For example, the multiple instances of the machine-learned model can be parallelized on a single processing device or computing device or parallelized across multiple processing devices or computing devices.

Each computing device that implements the machine-learned model or other aspects of the present disclosure can include a number of hardware components that enable performance of the techniques described herein. For example, each computing device can include one or more memory devices that store some or all of the machine-learned model. For example, the machine-learned model can be a structured numerical representation that is stored in memory. The one or more memory devices can also include instructions for implementing the machine-learned model or performing other operations. Example memory devices include RAM, ROM, EEPROM, EPROM, flash memory devices, magnetic disks, etc., and combinations thereof.

Each computing device can also include one or more processing devices that implement some or all of the machine-learned model and/or perform other related operations. Example processing devices include one or more of: a central processing unit (CPU); a visual processing

unit (VPU); a graphics processing unit (GPU); a tensor processing unit (TPU); a neural processing unit (NPU); a neural processing engine; a core of a CPU, VPU, GPU, TPU, NPU or other processing device; an application specific integrated circuit (ASIC); a field programmable gate array (FPGA); a co-processor; a controller; or combinations of the processing devices described above. Processing devices can be embedded within other hardware components such as, for example, an image sensor, accelerometer, etc.

Hardware components (e.g., memory devices and/or processing devices) can be spread across multiple physically distributed computing devices and/or virtually distributed computing systems.

In some implementations, the machine-learned models described herein can be trained at a training computing system and then provided for storage and/or implementation at one or more computing devices, as described above. For example, a model trainer can be located at the training computing system. The training computing system can be included in or separate from the one or more computing devices that implement the machine-learned model. As one example, Figure 4 illustrates a block diagram of an example computing device in communication with an example training computing system that includes a model trainer.

In some implementations, the model can be trained in an offline fashion or an online fashion. In offline training (also known as batch learning), a model is trained on the entirety of a static set of training data. In online learning, the model is continuously trained (or re-trained) as new training data becomes available (e.g., while the model is used to perform inference).

In some implementations, the model trainer can perform centralized training of the

machine-learned models (e.g., based on a centrally stored dataset). In other implementations, decentralized training techniques such as distributed training, federated learning, or the like can be used to train, update, or personalize the machine-learned models.

The machine-learned models described herein can be trained according to one or more of various different training types or techniques. For example, in some implementations, the machine-learned models can be trained using supervised learning, in which the machine-learned model is trained on a training dataset that includes instances or examples that have labels. The labels can be manually applied by experts, generated through crowd-sourcing, or provided by other techniques (e.g., by physics-based or complex mathematical models). In some implementations, if the user has provided consent, the training examples can be provided by the user computing device. In some implementations, this process can be referred to as personalizing the model.

As one example, Figure 5 illustrates a block diagram of an example training process in which a machine-learned model is trained on training data that includes example input data that has labels. Training processes other than the example process depicted in Figure 5 can be used as well.

In some implementations, the machine-learned model can be trained using example usage data from one or more example datasets. For instance, the training data can include example datasets and example usage of the datasets, such as example users, example interactions, example schedules of the users, and/or examples of any other data which may be used as input data during inference time by the model. The example datasets may be labelled with a known

(e.g. manually determined) likelihood that a portion of the dataset will be opened within a time period. The examples can be derived from real-world usage of the example datasets or can be prepared specifically for training the model.

In some implementations, the machine-learned model can be trained by optimizing an objective function. For example, in some implementations, the objective function can be or include a loss function that compares (e.g., determines a difference between) output data generated by the model from the training data and labels (e.g., ground-truth labels) associated with the training data. For example, the loss function can evaluate a sum or mean of squared differences between the output data and the labels. As another example, the objective function can be or include a cost function that describes a cost of a certain outcome or output data. Other objective functions can include margin-based techniques such as, for example, triplet loss or maximum-margin training.

One or more of various optimization techniques can be performed to optimize the objective function. For example, the optimization technique(s) can minimize or maximize the objective function. Example optimization techniques include Hessian-based techniques and gradient-based techniques, such as, for example, coordinate descent; gradient descent (e.g., stochastic gradient descent); subgradient methods; etc. Other optimization techniques include black box optimization techniques and heuristics.

In some implementations, backward propagation of errors can be used in conjunction with an optimization technique (e.g., gradient based techniques) to train a model (e.g., a multi-layer model such as an artificial neural network). For example, an iterative cycle of propagation

and model parameter (e.g., weights) update can be performed to train the model. Example backpropagation techniques include truncated backpropagation through time, Levenberg-Marquardt backpropagation, etc.

In some implementations, the machine-learned models described herein can be trained using unsupervised learning techniques. Unsupervised learning can include inferring a function to describe hidden structure from unlabeled data. For example, a classification or categorization may not be included in the data. Unsupervised learning techniques can be used to produce machine-learned models capable of performing clustering, anomaly detection, learning latent variable models, or other tasks.

In some implementations, the machine-learned models described herein can be trained using semi-supervised techniques which combine aspects of supervised learning and unsupervised learning.

In some implementations, the machine-learned models described herein can be trained or otherwise generated through evolutionary techniques or genetic algorithms.

In some implementations, the machine-learned models described herein can be trained using reinforcement learning. In reinforcement learning, an agent (e.g., model) can take actions in an environment and learn to maximize rewards and/or minimize penalties that result from such actions. Reinforcement learning can differ from the supervised learning problem in that correct input/output pairs are not presented, nor sub-optimal actions explicitly corrected.

In some implementations, one or more generalization techniques can be performed during training to improve the generalization of the machine-learned model. Generalization techniques

can help reduce overfitting of the machine-learned model to the training data. Example generalization techniques include dropout techniques; weight decay techniques; batch normalization; early stopping; subset selection; stepwise selection; etc.

In some implementations, the machine-learned models described herein can include or otherwise be impacted by a number of hyperparameters, such as, for example, learning rate, number of layers, number of nodes in each layer, number of leaves in a tree, number of clusters; etc. Hyperparameters can affect model performance. Hyperparameters can be hand selected or can be automatically selected through application of techniques such as, for example, grid search; black box optimization techniques (e.g., Bayesian optimization, random search, etc.); gradient-based optimization; etc. Example techniques and/or tools for performing automatic hyperparameter optimization include Hyperopt; Auto-WEKA; Spearmint; Metric Optimization Engine (MOE); etc.

In some implementations, various techniques can be used to optimize and/or adapt the learning rate when the model is trained. Example techniques and/or tools for performing learning rate optimization or adaptation include Adagrad; Adaptive Moment Estimation (ADAM); Adadelta; RMSprop; etc.

In some implementations, transfer learning techniques can be used to provide an initial model from which to begin training of the machine-learned models described herein.

In some implementations, the machine-learned models described herein can be included in different portions of computer-readable code on a computing device. In one example, the machine-learned model can be included in a particular application or program and used (e.g.,

exclusively) by such particular application or program. Thus, in one example, a computing device can include a number of applications and one or more of such applications can contain its own respective machine learning library and machine-learned model(s).

In another example, the machine-learned models described herein can be included in an operating system of a computing device (e.g., in a central intelligence layer of an operating system) and can be called or otherwise used by one or more applications that interact with the operating system. In some implementations, each application can communicate with the central intelligence layer (and model(s) stored therein) using an application programming interface (API) (e.g., a common, public API across all applications).

In some implementations, the central intelligence layer can communicate with a central device data layer. The central device data layer can be a centralized repository of data for the computing device. The central device data layer can communicate with a number of other components of the computing device, such as, for example, one or more sensors, a context manager, a device state component, and/or additional components. In some implementations, the central device data layer can communicate with each device component using an API (e.g., a private API).

The technology discussed herein makes reference to servers, databases, software applications, and other computer-based systems, as well as actions taken and information sent to and from such systems. The inherent flexibility of computer-based systems allows for a great variety of possible configurations, combinations, and divisions of tasks and functionality between and among components. For instance, processes discussed herein can be implemented

using a single device or component or multiple devices or components working in combination. Databases and applications can be implemented on a single system or distributed across multiple systems. Distributed components can operate sequentially or in parallel.

In addition, the machine learning techniques described herein are readily interchangeable and combinable. Although certain example techniques have been described, many others exist and can be used in conjunction with aspects of the present disclosure.

Thus, while the present subject matter has been described in detail with respect to various specific example implementations, each example is provided by way of explanation, not limitation of the disclosure. One of ordinary skill in the art can readily make alterations to, variations of, and equivalents to such implementations. Accordingly, the subject disclosure does not preclude inclusion of such modifications, variations and/or additions to the present subject matter as would be readily apparent to one of ordinary skill in the art. For instance, features illustrated or described as part of one implementation can be used with another implementation to yield a still further implementation.

A brief overview of example machine-learned models and associated techniques has been provided by the present disclosure. For additional details, readers should review the following references: *Machine Learning A Probabilistic Perspective* (Murphy); *Rules of Machine Learning: Best Practices for ML Engineering* (Zinkevich); *Deep Learning* (Goodfellow); *Reinforcement Learning: An Introduction* (Sutton); and *Artificial Intelligence: A Modern Approach* (Norvig).

Figures

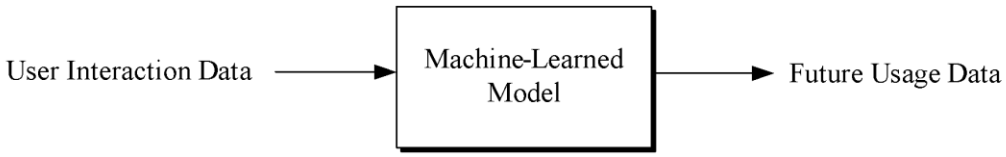

Figure 1

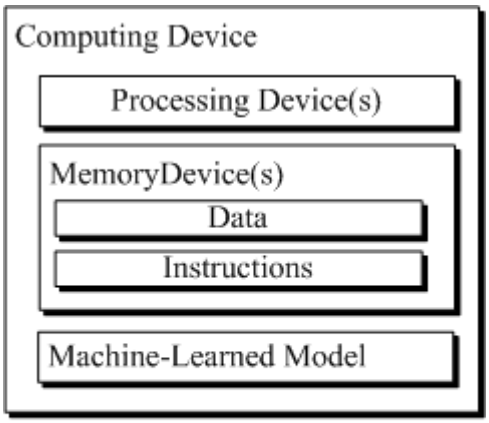

Figure 2

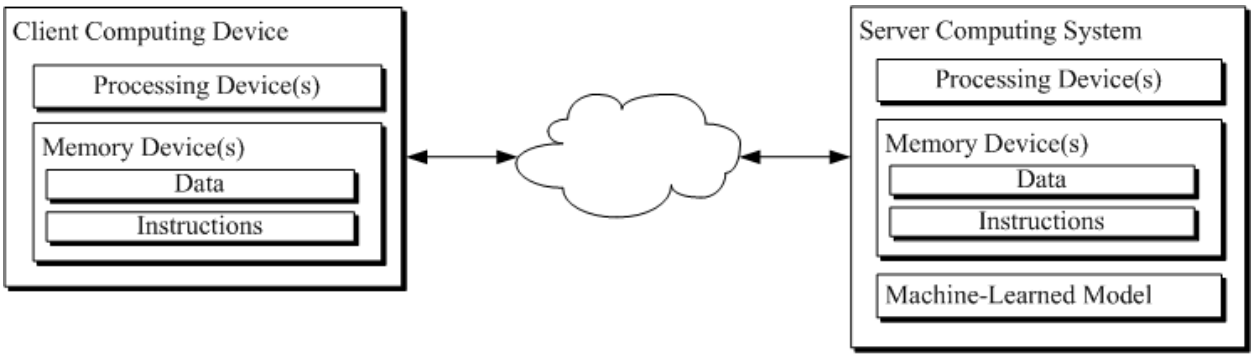

Figure 3

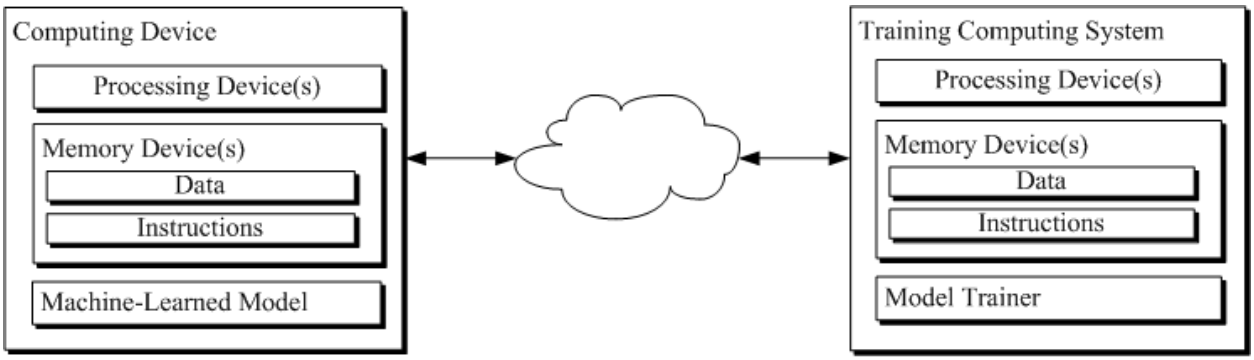

Figure 4

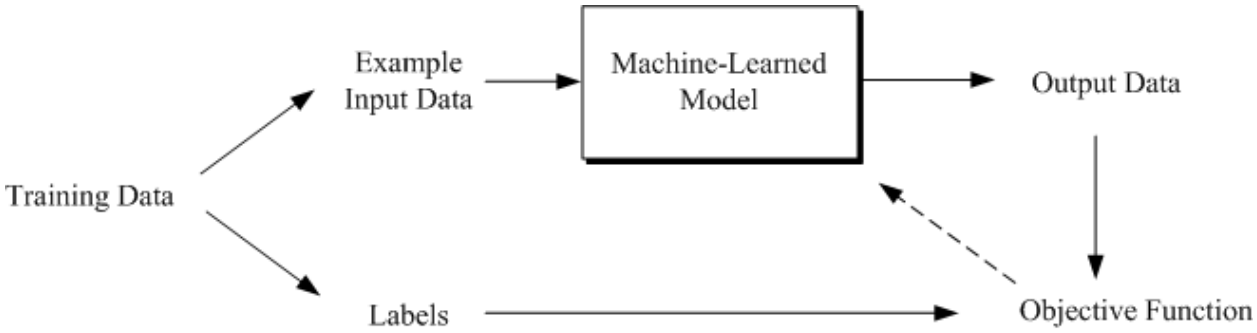

Figure 5

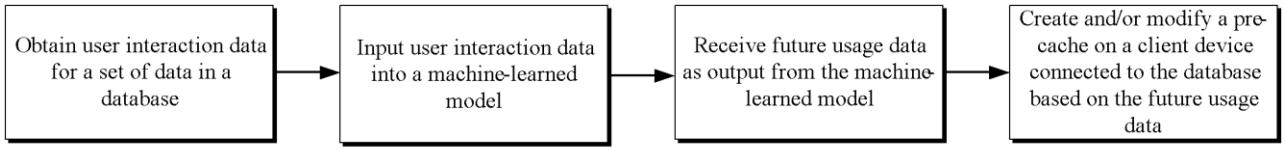

Figure 6